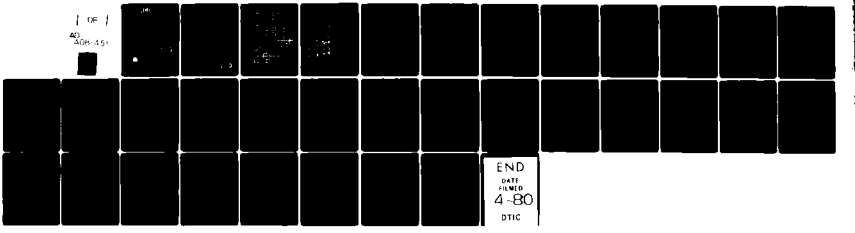
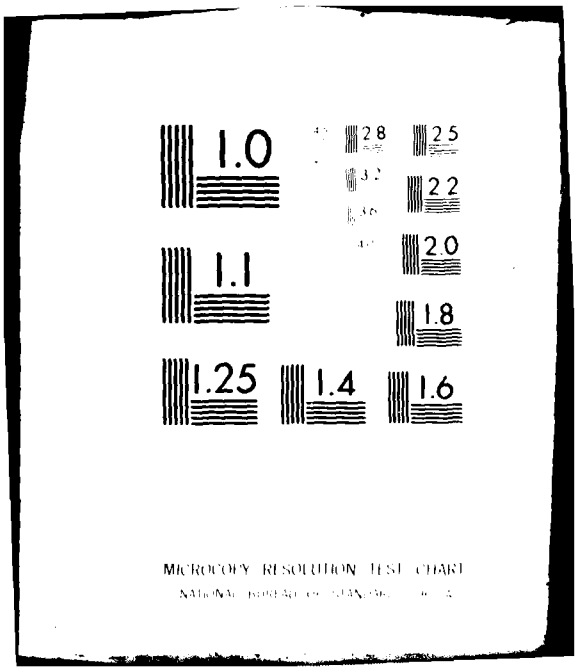
AD-A081 451 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/3
DESIGN OF SPECIAL-PURPOSE VLSI CHIPS: EXAMPLE AND OPINIONS.(U)
SEP 79 H J FOSTER, H T KUNG N00014-76-C-0370
UNCLASSIFIED CMU-CS-79-147 NL




1.0
2.8
2.5
3.2
2.2
3.6
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

(12) LEVEL II

CMU-CS-79-147

B.S.

Design of Special-Purpose VLSI Chips: Example and Opinions

M. J. Foster and H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

September 1979

AD A081451

DEPARTMENT
of
COMPUTER SCIENCE

DTIC
ELECTE
MAR 6 1980
S D
B

DDC FILE COPY

Carnegie-Mellon University

80 3 5 010

CMU-CS- 79-147

-9

Design of Special-Purpose VLSI Chips: Example and Opinions

M. J. Foster and H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

September 1979

This research was supported in part by the Defense Advanced Research Projects Agency under Contract F33615-78-C-1551 (monitored by the Air Force Office of Scientific Research), the National Science Foundation under Grant MCS 78-236-76, the Office of Naval Research under Contract N00014-76-C-0370, and an NSF Fellowship.

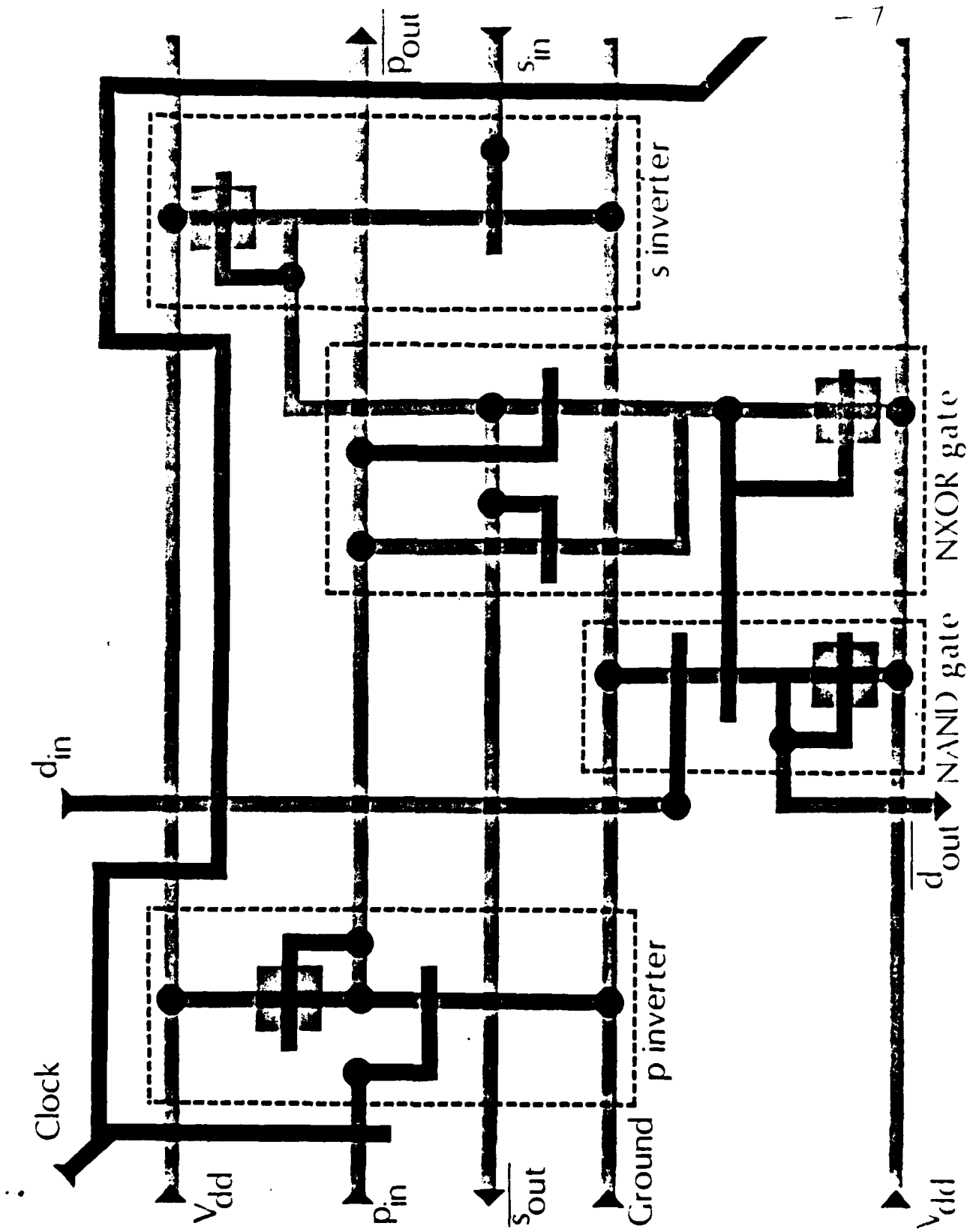


Plate 1: Stick diagram for the positive comparator cell.

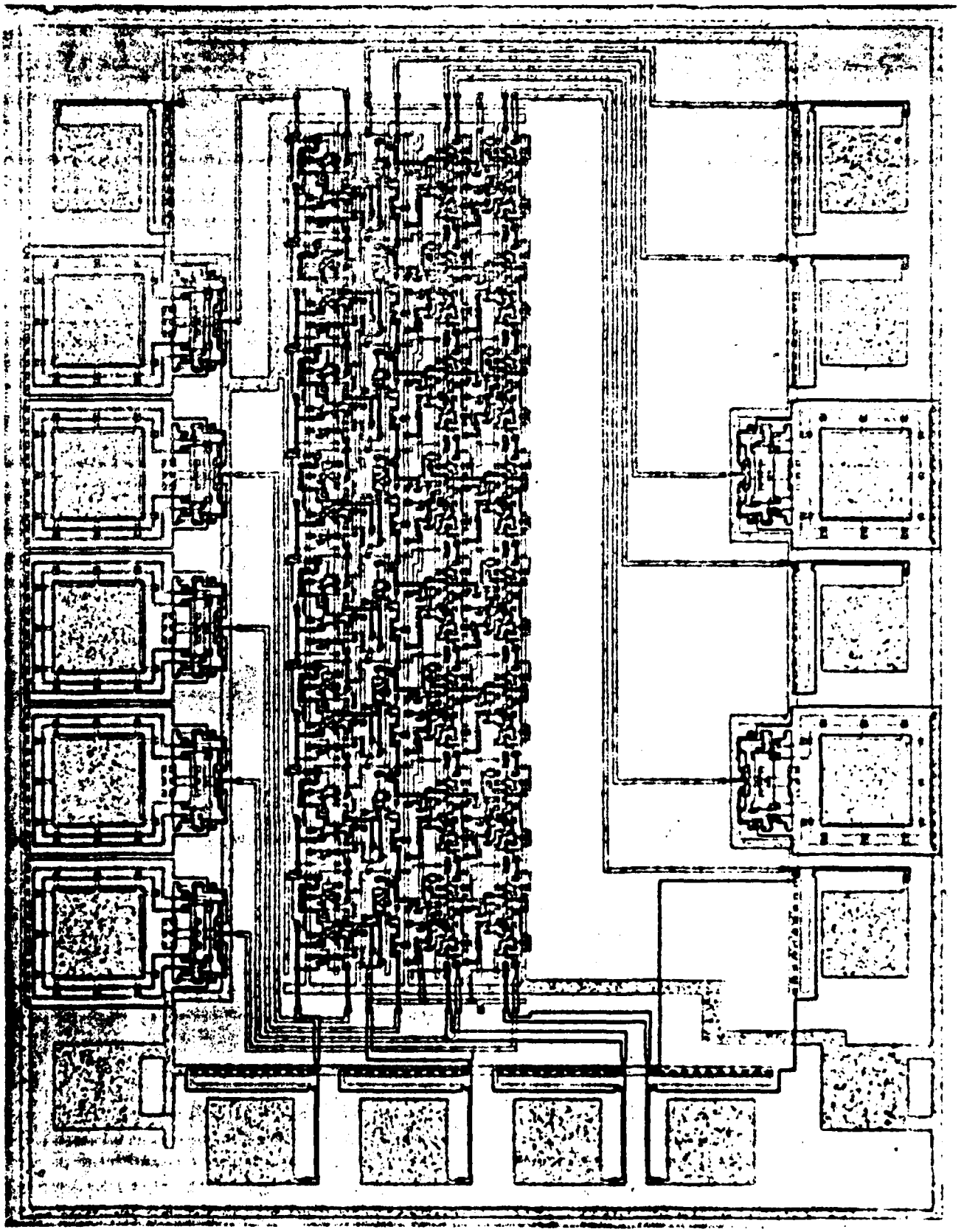


Plate 2: The pattern matching chip.

ABSTRACT

This paper identifies important steps in the design of a special purpose VLSI chip, and argues that the most crucial step is the design of the underlying algorithm. Because the algorithm determines the degree of parallelism and pipelining that is possible, it largely determines the performance of the chip. Furthermore, if the underlying algorithm has the right properties such as modularity and regularity, then the rest of the design should be routine and thus takes little effort. These claims are supported by a concrete example -- the design of an efficient pattern matching chip, which has been fabricated for testing.

Table of Contents

1. Introduction

We have now entered a technological domain where many of the problems encountered before in building special-purpose hardware are no longer so severe. Current LSI technology allows tens of thousands of devices to fit on a single chip, and future advances should increase this number. Devices whose construction used to require many components can now be built with just a few chips. This reduces the difficulties in, for example, the reliability, performance, and heat dissipation problems that arise from combining many standard SSI or MSI components. Furthermore, although design of integrated circuits has been regarded as difficult, the development of simplified techniques for structuring IC system design such as those of [Mead and Conway 80] together with implementation guides such as [Hon and Sequin 79] allow relatively naive designers to achieve success.

Special-purpose VLSI chips can be used as peripheral devices attached to a conventional host computer. The resulting system can be considered as an efficient general-purpose computer, if many types of chips are attached. Figure 1-1 illustrates how special-purpose chips such as the pattern matcher, FFT device, and sorter might form a part of a general-purpose computer system.

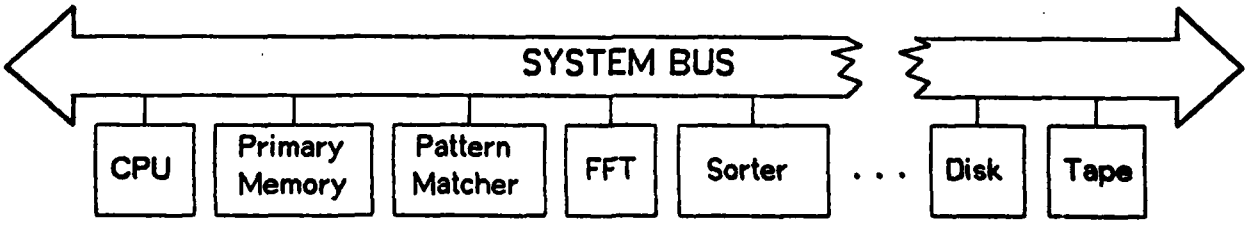


Figure 1-1: Special-purpose chips attached to a general-purpose computer.

Although it is now feasible to construct complex special-purpose chips, the approach can become successful only if the cost of designing these chips is not prohibitive. We argue in Section 2 that the chip design time, which constitutes the major design cost, can be reduced significantly if the underlying algorithm is designed carefully in the first place. "Good" algorithms are characterized in that section. This design philosophy is supported in Section 3 by a concrete example -- the design of a pattern matching chip. This chip was designed by the authors in the Spring of 1979 at CMU. Prototype chips have been fabricated, and are in

the process of testing. Preliminary results show that the chip can achieve a data rate of one character every 250 ns, which is higher than the memory bandwidth of most conventional computers. This high performance is achieved in spite of little effort given to the circuit and layout design. We attribute the performance mainly to the careful design of the underlying algorithm.

Section 4 identifies the major steps in designing a special purpose VLSI chip. A methodology is given that transforms an algorithm design into its final layout design in a more or less mechanical way. We conclude in Section 5 that the time to construct special purpose chips has come; with the right design philosophy and methodology, designing a special purpose chip should not be more difficult than designing a high level algorithm for the same job!

2. Design Philosophy

We believe that in the course of designing a VLSI special purpose chip the most crucial decision is the choice of the underlying algorithm. A large portion of the design effort should go into designing and refining the algorithm, before beginning its realization at the circuit or layout level.

Algorithms that perform well on conventional random access computers are not always the best for VLSI implementation. As pointed out in [Sutherland and Mead 77], good algorithms for VLSI implementation are not necessarily those requiring minimal computation. Computation is cheap in VLSI; communication determines the performance. Thus in this new era of computation we need to reconsider the algorithms for many tasks.

The characteristics of a good algorithm

A "good" algorithm in this context should possess one or more of the following properties:

1. The algorithm can be implemented by only a few different types of simple cells.
2. The data and control flow of the algorithm is simple and regular, so that in the implementation cells can be connected by a network with local and regular interconnections. Long distance or irregular communication is thus minimized.
3. The algorithm uses extensive pipelining and multiprocessing. Typically, several data streams move at constant velocity over fixed paths in the network, interacting at cells where they meet. In this way a large number of cells are active at one time so that the computation speed can keep up with the data rate.

Algorithms with these properties have been named systolic algorithms[1] in [Kung and Leiserson 79]. Many systolic algorithms have been designed recently; see the survey given in [Kung 79a].

Advantages of using good algorithms

Most special-purpose chips will be made in relatively small quantities, so the design cost must be kept low. Several aspects of systolic algorithms help ease the design task:

1. One has to design and test only a few different, simple cells, as most of the cells on a chip are copies of a few basic ones.
2. Regular interconnection implies that the design can be made modular and

[1] The word "systole" was borrowed from physiologists who use it to refer to the rhythmically recurrent contractions of the heart and arteries which pulse blood through the body. For a systolic algorithm, the function of a cell is analogous to that of the heart or arteries. Each cell regularly pumps data in and out, each time performing some short computation, so that a regular flow of data is kept up in the network.

extensible. A large chip can be designed by combining the designs of small chips.

3. By pipelining and multiprocessing the performance requirement of a special purpose chip can be met by simply including many identical cells on the chip.

All these imply that if a good algorithm is used, the design time, and therefore the design cost, can be substantially reduced.

Summary of the design philosophy

Design modularity should be enforced at the algorithm design level. Performance and design cost are determined, to a great extent, by the algorithm. A large portion of the design time should therefore be devoted to algorithm design. Low level optimizations at the circuit or layout design level are probably not worthwhile in most cases, as these will only lead to minor improvements in the overall performance, but may greatly increase the design time.

3. The Design of a Pattern Matching Chip

This section describes the design of a specific VLSI chip. The chip that we chose to illustrate our design philosophy and methodology performs on-line pattern matching of strings with wild card characters. We discuss the design of the underlying algorithm, and demonstrate that it can be mapped to circuit and layout designs in a straightforward way.

This section can be read independently from the other sections as a detailed description of the process of designing a chip in NMOS. The reader who is about to design his or her first chip may find this section instructive.

3.1 The String Pattern Matching Problem

Our chip accepts two streams of characters from the host machine, and produces a stream of bits as shown in Figure 3-1. One of the input streams, called the text string, is an endless string of characters over some alphabet Σ. The other input stream, the pattern, contains a fixed length vector of characters over the alphabet $\Sigma \cup \{x\}$, where x is the wild card character. The output is a stream of bits, each of which corresponds to one of the characters in the text string. The data streams move at a steady rate between the host computer and the pattern matcher, with a constant time between data items.

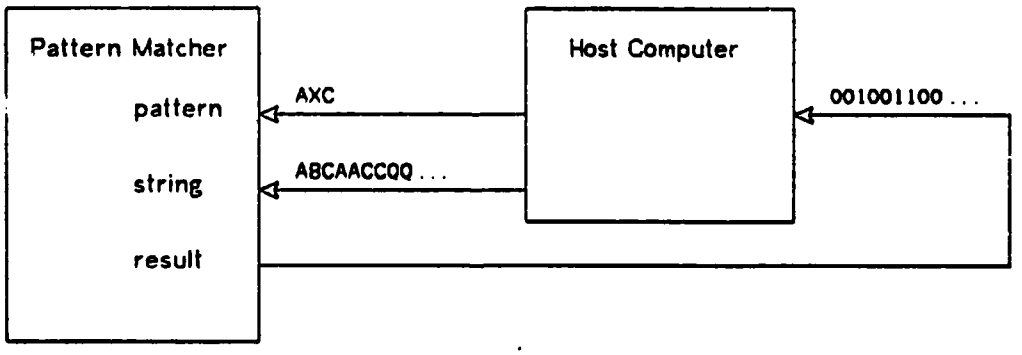


Figure 3-1: Data to and from the pattern matcher.

Let us denote the input text stream as $s_0 s_1 s_2 \ldots$. The finite pattern stream will be denoted as $p_0 p_1 \ldots p_k$, and the output result stream as $r_0 r_1 r_2 \ldots$. Characters in the two input streams

may be tested for equality, with the wild card character x deemed to match any character in Σ. The output bit r_i is to be set to 1 if the substring $s_{i-k}s_{i+1-k}...s_i$ matches the pattern, and 0 otherwise. That is,

$$r_i \leftarrow (s_{i-k} = p_0) \wedge (s_{i+1-k} = p_1) \wedge \ldots \wedge (s_i = p_k).$$

In Figure 3-1, for example, the pattern AXC matches the substrings $s_0s_1s_2$, $s_3s_4s_5$, and $s_4s_5s_6$ (ABC, AAC, and ACC). Result bits r_2, r_5, and r_6 are thus set to 1, and all other result bits are 0.

This problem is important in many applications. String pattern matching is a basic operation in SNOBOL-like languages [Griswold et al. 68] and in database query languages. String matching hardware has been proposed for use in office automation systems [Warter and Mules 79]. Many artificial intelligence systems make heavy use of pattern matching as a method of search. Furthermore, as pointed out in Section 3.4 below, string pattern matching is similar in form to many stressing numerical computations, such as computing convolutions and correlations. All of the linear product problems discussed in [Fischer and Paterson 74] are similar to string matching.

Several fast algorithms are known for solving the string matching problem without wild card characters on a normal random access machine [Boyer and Moore 77, Knuth et al. 77]. These methods use information about partial matches of the pattern with itself to avoid redundant comparisons, skipping over parts of the string where partial match results may be inferred from previous comparisons. When wild card characters exist in the pattern these methods break down, since the "matches" relation is no longer transitive. The strings AC and XB both match AX, for example, but do not match each other. Information about matchings of the pattern with itself is therefore irrelevant if wild card characters are present. The fastest algorithm known for string matching with wild card characters is based on multiplication of large integers [Fischer and Paterson 74], and requires more than linear time. The pattern matching chip solves the problem in linear time by performing comparisons in parallel.

3.2 The Chip Design

3.2.1 Algorithm Design

Data flow

The pattern and the text string arrive alternately over the bus one character at a time. We will call the interval during which one character arrives from either stream a beat. During each pair of consecutive beats the chip must input two characters and output one result. All

characters on the chip move during each beat.

The chip is divided into character cells, each of which can compare two characters and accumulate a temporary result. The pattern and string follow a preset path of cells from the time they enter the chip until the time they leave it. On each beat every character moves to a new cell. We use a linear array of cells, with the pattern and string moving in opposite directions, to make each character of the string move past all characters of the pattern. To make each pair of characters meet, rather than just pass, we must separate them by one cell so that alternate cells are idle. Each cell is then active on alternate beats. The flow of characters is traced for several beats in Figure 3-2.

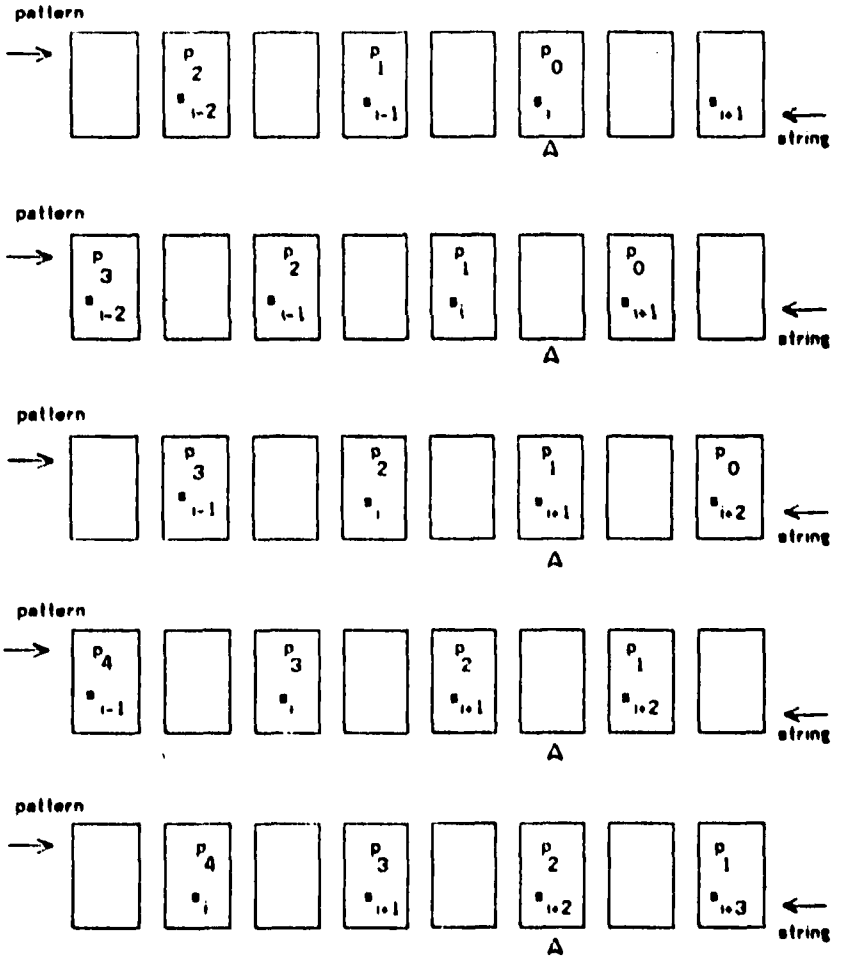


Figure 3-2: The flow of characters.

For illustration let us follow the history of the character cell indicated by the arrowhead in Figure 3-2, starting when the first character of the pattern, p_0, is present. Suppose the string character s_i is present during this beat. During the next beat the cell is idle, but during the beat after that it contains p_1 and s_{i+1}. Two beats later, p_2 and s_{i+2} are together, then p_3 and s_{i+3}, and so on. By the time the last pattern character p_k leaves the cell, the substring $s_i s_{i+1} \ldots s_{i+k}$ will have met the whole pattern. We can therefore keep the partial match results in this cell, update it whenever a new pair of characters enters the cell, and output the results after the last character of the pattern goes past. To output results we shift them along with the string, so that each match result leaves the array with the last character of its substring. If we recirculate the pattern so that the first character follows two beats after the last one, we can output the completed result and initialize a new partial result on the same beat. The number of character cells required is therefore no more than the number of characters in the pattern.

Each character cell performs two separate functions: it compares characters of the pattern and string, and it updates and outputs match results. We can divide these functions between two modules, so that there are two linear arrays with connections between corresponding cells as shown in Figure 3-3. The cells on the top are the comparators; the pattern flows through them from left to right, and the string flows from right to left. The bottom cells, or accumulators, receive the results of the comparison from above. They maintain partial results, and shift completed results right to left. Two bits associated with the pattern flow through the accumulators from left to right. One of these bits, called λ, marks the end of the pattern. It is one for the last character of the pattern and zero for the others. The other bit is x, the don't care bit, which marks the wild card characters. A one in this bit tells the accumulator to ignore the result from the comparator, since this pattern character matches anything.

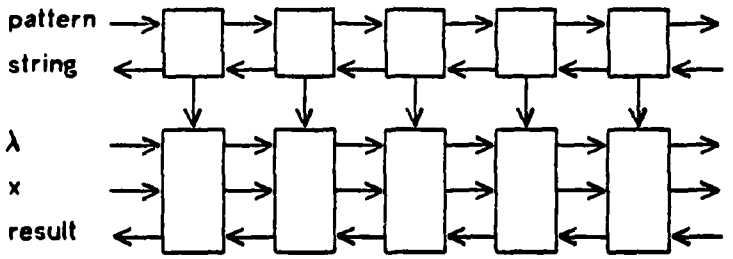


Figure 3-3: Comparators (on the top) and accumulators (on the bottom.)

We can further divide the comparators. Rather than using one large circuit to compare whole characters, we can divide each comparator into modules that can compare single bits. Two characters are equal if corresponding bits are equal. By staggering the bits so the high order bits enter the array before the low order ones, we can make a pipeline comparator. Each single bit comparator shifts its result down to meet the bits coming into the next lower comparator. The active and idle comparators alternate vertically as well as horizontally, so that on each beat the active comparators form a checkerboard pattern as shown in Figure 3-4.

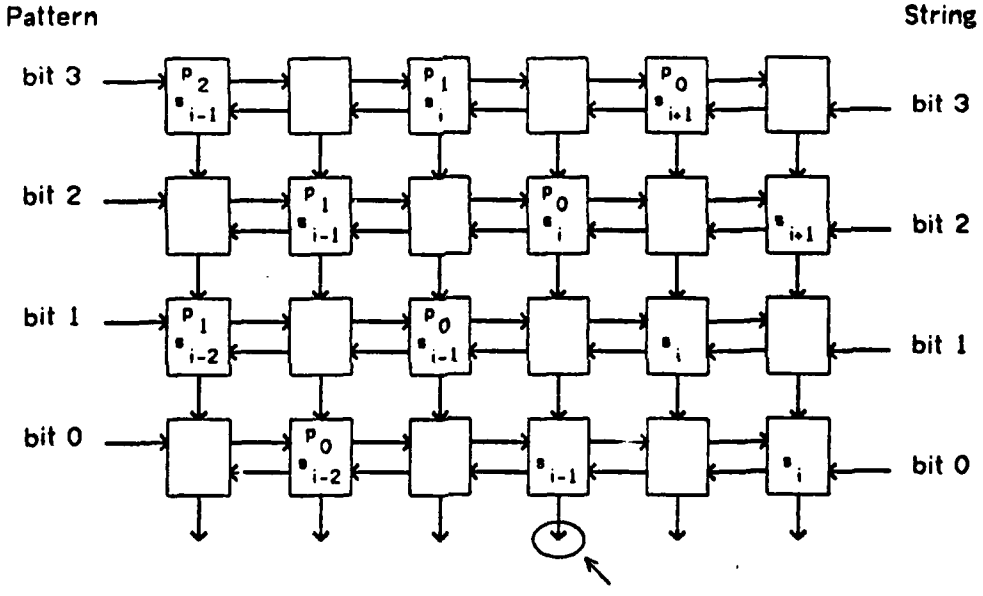


Figure 3-4: Comparators for single bits.

Cell algorithms

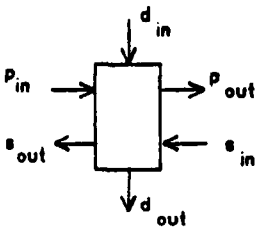
Now we are ready to design the two kinds of cells to be used in the pattern matcher.

- The <u>one-bit comparator</u> has one bit of the pattern flowing from left to right, one bit of the string flowing from right to left, and the comparison result for the pair of characters flowing from top to bottom. The cell uses this algorithm to update the comparison result:

$p_{out} \leftarrow p_{in}$

$s_{out} \leftarrow s_{in}$

$d_{out} \leftarrow d_{in}$ AND ($p_{in} = s_{in}$)



- The accumulator receives d_{in}, the result from the comparator above, λ_{in}, the end of pattern indicator, and x_{in}, the don't care bit. It maintains a temporary result t, and at the end of the pattern uses t to replace the result r that flows from right to left.

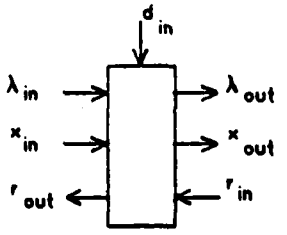
$\lambda_{out} \leftarrow \lambda_{in}$

$x_{out} \leftarrow x_{in}$

IF λ_{in}

THEN $r_{out} \leftarrow t$; $t \leftarrow$ TRUE

ELSE $r_{out} \leftarrow r_{in}$; $t \leftarrow t$ AND (x_{in} OR d_{in})



3.2.2 Circuit and Layout Design

Dataflow circuit

Each of the pipelines used for data flow in the algorithm is implemented as a unidirectional shift register that shifts on each beat. Every other cell of the shift register contains valid data. In NMOS, the technology used for this chip, a shift register is composed of a chain of inverters separated by pass transistors as shown in Figure 3-5. When the voltage on the gate of a transistor is near the supply voltage V_{dd}, its channel conducts current, while if the voltage is near ground it does not. The inputs to the inverters can store charge, so data is stored within the inverters, and the pass transistors control the inverter inputs. A clock with two non-overlapping phases controls the pass transistors. Adjacent transistors are turned on by opposite phases of the clock, so that there is never a closed path between inverters that are separated by two transistors. Alternate inverters can therefore store independent data bits.

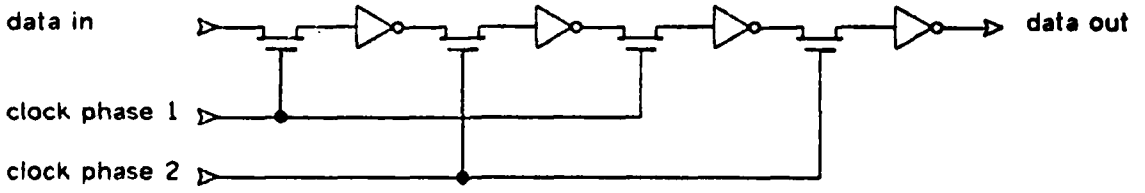


Figure 3-5: A shift register in NMOS.

The dynamic alternation of active and idle inverters in the NMOS shift register mirrors the alternation of active and idle cells in the algorithm (cf. Figure 3-4). Each cell can thus contain one gated inverter from each of the shift registers that passes through it. The clock controlling the shift register stages in a cell can activate the cell. The shift register components are then fully utilized: all idle inverters are in idle stages.

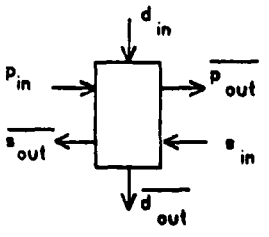
Cell circuit

Since each cell inverts its inputs before sending them to its neighbors, two versions of each cell must be constructed. One version operates on positive inputs to produce inverted outputs, while the other computes positive outputs from inverted inputs. Transforming a cell algorithm to its inverted twin is straightforward, so the existence of two versions presents no problems. Using the cell algorithms, we can design circuits for the twin versions of each cell. From the circuit designs, we can lay out the masks for fabricating the chip. We will illustrate the process with the positive version of the comparator cell. This version takes positive inputs and produces inverted outputs, so we must invert the outputs in the comparator algorithm:

$$\overline{p_{out}} \leftarrow \text{NOT } p_{in}$$

$$\overline{s_{out}} \leftarrow \text{NOT } s_{in}$$

$$\overline{d_{out}} \leftarrow d_{in} \text{ NAND } (p_{in} = s_{in})$$



In NMOS, data storage can take place on the input to any logic gate, as long as that input can be isolated by a pass transistor. We will implement the p and s shift registers with

inverters, as planned, but we can use a NAND gate as the stage for the d shift register. Figure 3-6 is the circuit for the positive comparator. When the clock input goes from ground to V_{dd}, the power supply voltage, all three pass transistors turn on. The pattern and string inputs are then stored on the inverters, and the d input is stored on one input to the NAND gate. The exclusive NOR gate outputs TRUE if the two inputs are equal, and FALSE otherwise. The output of this equality test goes to the other input of the NAND gate, which computes $\overline{d_{out}}$. After the inputs have stabilized, the clock goes to ground. The outputs of this cell then provide stable inputs to neighboring cells until the clock goes high once again.

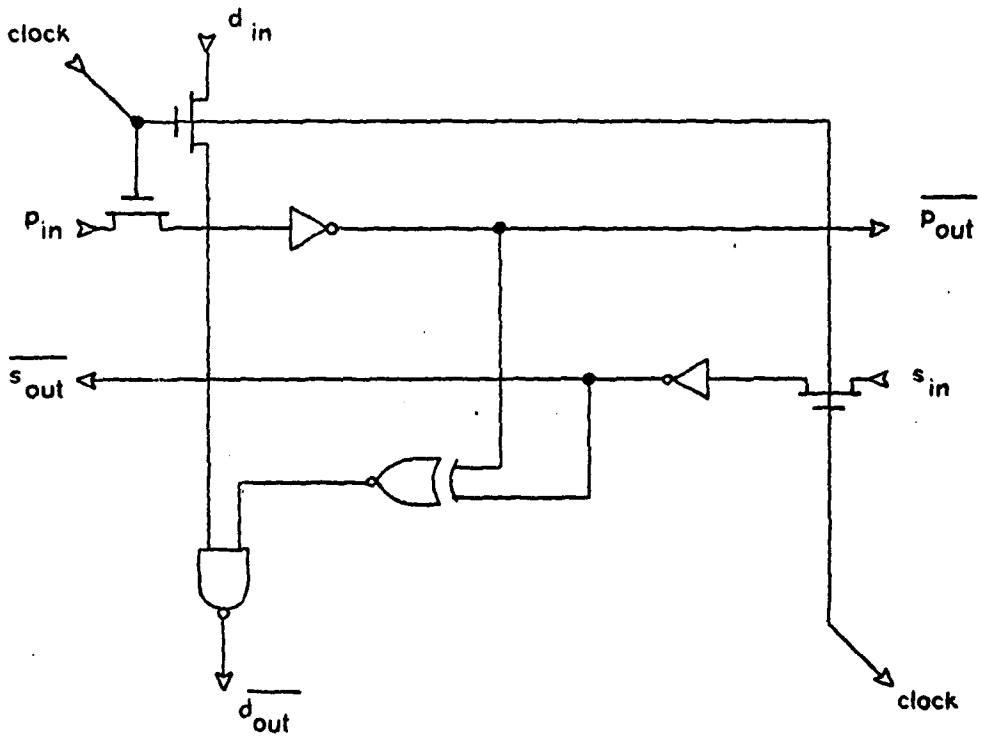


Figure 3-6: Positive comparator circuit.

Cell sticks

The next step after completing the circuit diagram is to design the topological layout, or stick diagram, for the cell. The stick diagram shows the relative positions of all signal paths, power connections, and components, but hides their absolute sizes and positions. Most of the circuit components can be implemented in several ways, and a choice among these

implementations must be made during this stage of the design.

Silicon-gate NMOS technology uses three conduction layers, which are differentiated by color in the stick diagram. Following the convention in [Mead and Conway 80], in our diagrams blue lines represent metal conduction paths, red lines represent polycrystalline silicon (polysilicon) and green lines represent diffusion into the substrate. The three layers are insulated from each other except at contact cuts, which are represented by round black dots. The yellow squares are areas of ion implantation, used to create depletion mode transistors. These serve as pullup resistors in the gates and inverters.

Field-effect transistors are created in NMOS by crossing a diffusion path (green) with a polysilicon area (red). The green path is the channel, and the red area is the gate. If no ion implantation is present, the channel conducts current only when the gate is at V_{dd}.

The positive comparator cell uses pass transistors and inverters to implement the shift registers. In addition to these, it uses a NAND gate and an equality, or NXOR, gate. These basic components are combined in Plate 1 to produce the stick diagram for the positive comparator cell. Power and ground run horizontally across the cell on metal (blue) paths. The clock is in polysilicon (red) at the top and right edges of the cell. It dips below the upper power wire near the middle of the cell to allow the cell above to connect to the power wire. Data paths for p and s run horizontally along the top, while d runs downward in diffusion (green).

Let us trace the p data path through the cell. It enters at the left in diffusion, and passes through the channel of a transistor that is gated by the clock. Contact is made to a polysilicon path that goes to the input of the p inverter. The inverter output, in metal, crosses the d data path with no interaction and provides an input to the equality gate. It then passes over the s inverter, and leaves the cell at the right.

Layout

When stick diagrams have been designed for all of the cells, actual layouts can be produced. These layouts follow the topology of the stick diagrams, but also include the absolute sizes and positions of all components. Designing a layout involves choosing electrical parameters for all transistors, as well as following minimum spacing rules for the intended fabrication process. Care must be taken to line up power connections and data paths that cross several cells. In principle the layout can be designed mechanically from the circuit and stick diagrams.

When the layouts for all cells are complete, they are assembled into a working array with the inputs and outputs hooked to contact pads. Layouts are described using a graphics

language (such as Caltech Intermediate Form, see [Mead and Conway 80]) that can be interpreted to make the masks. These masks are then used to fabricate the chips. Plate 2 is a photograph of a prototype pattern matching chip that can handle patterns containing up to eight two-bit characters.

3.3 Discussion of Design Alternatives

There were many points in the design of the pattern matching chip where a choice among several alternatives was made. This section describes some of the more important design decisions. There were three major areas of choice: choice of an algorithm, choice of a data flow implementation, and choice of a method for cell implementation.

3.3.1 Alternative Algorithms

There is a bewildering variety of algorithms that could form the basis for a pattern matching chip. The desire for simple and regular data flow rules out the fast sequential algorithms described in [Knuth et al. 77] and [Boyer and Moore 77]. It seems that these algorithms require dynamically changing communication, so that any hardware implementation will not be modular and is likely to be quite complex.

Mukhopadhyay [Mukhopadhyay 79] has proposed several machines in which each cell stores a character of the pattern, and the text string is broadcast character by character to all cells. The broadcast communication is the major disadvantage of this algorithm. Each cell requires a connection to the broadcast channel, which either increases the power requirements of the system as a whole or decreases its speed. Our algorithm requires no broadcasting of data.

An algorithm that is similar to ours uses a linear array of cells with data flowing in only one direction. The pattern is permanently stored in the array of cells, and the text string moves past it. Partial results move at half the speed of the text so that they accumulate results from an entire substring match. This algorithm was rejected because of the static storage of the pattern. Loading the cells in preparation for a pattern match would require extra time and circuitry.

The algorithm actually chosen (described in Section 3.2.1) is well suited to VLSI implementation. All communication is local, since each character cell communicates only with its left and right neighbors. This local communication enhances modularity and extensibility, as well as avoiding the large drivers needed for long range transmission. Only a few types of cells are used, with many copies of each type. By replicating the basic cells, pattern matching chips of any size can be formed. Finally, there is no separate operation required to

set up the system for a new problem, so control of the chip is simplified.

3.3.2 Alternative Data Flow Implementations

Although the global flow of data is determined by the choice of algorithm, several methods of implementing the data flow may be possible. Serial or parallel data transmission between cells may be selected, for example. Communication may be coordinated in several ways. The data flow can even be transformed to combine several cells into one circuit. We will discuss two of the decisions that arose in implementing the data flow of the pattern matching chip.

The existence of idle cells can be avoided by combining pairs of neighboring cells when implementing the data flow. Each cell pair then contains one active cell and one idle cell at each beat, so circuitry can be shared between the two cells. For the pattern matching chip, for example, the equality gate could have been shared between neighboring comparators, and the d data path could have been multiplexed.

If the amount of circuitry that can be shared is large enough, it may be advantageous to combine two or more cells in this way. Some additional circuitry will of course be needed to coordinate the sharing, and this may wipe out the savings. The increased interdependence of the circuit components may also offset the savings, since design changes may become more difficult, and errors may be made. The pattern matcher cells are too small to profit from this data flow transformation.

Another choice in data flow implementation is between self-timed and clocked (synchronous) data paths. In a clocked data flow implementation, all data movement is under a centralized control. The data flow controller sends signals to each cell to enable data transfers. The pattern matching chip uses clocked data flow. In fact, the data flow control signals are the same clock signals needed for data refreshing, although this need not be true in general.

In a self-timed implementation, data flow control is distributed among the cells, so that each cell controls its own data transfers. Neighboring cells must obey a signalling convention to coordinate their communication. Self-timed data flow has advantages in modularity and extensibility, since no common clock is needed. Each of the cells may run at its own pace, synchronizing with its neighbors only when communication is needed. The disadvantage is the extra circuitry needed to implement the signalling conventions. For systems that are small enough to use a common clock, like the pattern matching chip, the clocked data flow implementation should be chosen. For larger systems, of course, self-timed communication may have to be used (see, e.g. [Seitz 79]).

3.3.3 Alternative Cell Implementations

Two major decisions affected the design of the cells. Static shift registers, which can hold data for long periods without shifting it, were rejected in favor of dynamic shift registers, which cannot. Also, a random logic implementation of the cell circuitry was chosen rather than a more structured approach using standard PLA (programmed logic array) and register layouts.

The dynamic shift registers that we used are incapable of holding data for more than about 1ms. without shifting. Data is refreshed only by shifting it. Static shift registers, the alternative choice, have regeneration circuitry in every stage so that data can be held indefinitely without shifting it. A third signal, in addition to the two clock phases, is needed to command the register to shift.

Static shift registers are probably the better choice for most systems. They do not invert data between stages, as do dynamic shift registers, and they simplify testing. For this chip, however, dynamic shift registers have advantages. The alternation of active and idle cells allows just one inverter from each shift register to be placed in each cell. This permits the two phase clock to do double duty as a data flow control signal. The cells and the global layout are thus simplified greatly.

The use of random logic was dictated by the simplicity of the cell functions. If cells contain more than a few gates the state-machine design approach should be taken. The state of the cell may be held in a register, and the combinational logic used for changing states can be implemented with a PLA. Standard layouts for registers and PLA cells are available, so this approach simplifies the design and layout tasks. It also shortens the time for design changes and correction of errors. The small size of the pattern matcher cells, which contain only four gates each, made the use of random logic possible. Design and layout of such simple circuits is quite easy.

3.4 Uses and Extensions of the Pattern Matching Chip

A pattern matching chip with n character cells can be used directly only for matching patterns of length up to n. Longer patterns require the existence of more than n partial results at each beat. Since any chip must be of finite size, it is important that the chip be extensible. It should be possible to combine several chips to form a larger pattern matcher.

In order to make the chip extensible, more inputs and outputs must be provided. Specifically, an input for the result stream and outputs for the pattern and text streams must

be available. Several pattern matching chips can then be cascaded, as shown in figure 3-7. The inputs to each chip in figure 3-7 are taken from the outputs of its neighbors, so that the cells on all of the chips form a single linear array. The pattern is fed to the inputs of the leftmost chip, and the text string is input to the rightmost chip. The result output is taken from the leftmost chip. A cascade of k chips with n cells each can match patterns of up to kn characters.

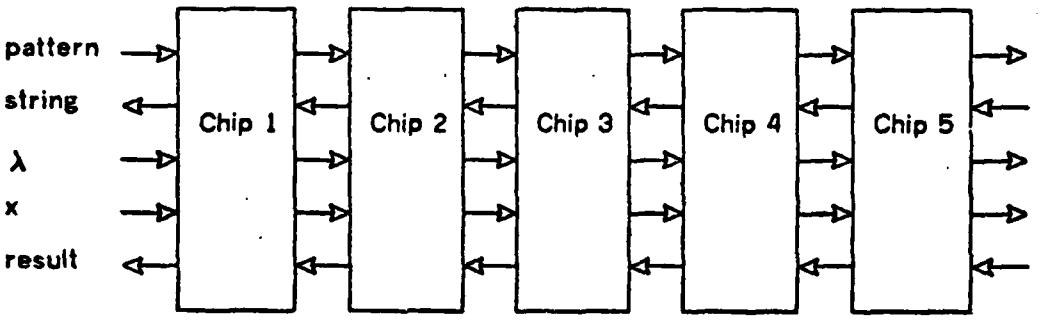


Figure 3-7: A five chip pattern matcher.

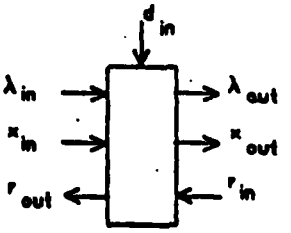
If the pattern to be matched is longer than the capacity of the available pattern matching system, the pattern can be run through the system several times to match it against the entire string. If the system contains a total of n character cells, each run will match the complete pattern against n substrings. To cover all substrings, all we need do is delay the string by n characters on succeeding runs.

Many problems other than string matching can be solved by similar algorithms. Special purpose hardware for these problems can be designed by modifying the design of the pattern matcher. For example, we might wish to count how many characters in each substring match the corresponding characters in the pattern. This problem can be solved by replacing the result bit stream by a stream of integers, and replacing the accumulator cell by a counting cell:

$\lambda_{out} \leftarrow \lambda_{in}$

$x_{out} \leftarrow x_{in}$

IF λ_{in}
  THEN $r_{out} \leftarrow t$; $t \leftarrow 0$
ELSE IF x_{in} OR d_{in}
  THEN $t \leftarrow t+1$; $r_{out} \leftarrow r_{in}$
ELSE $r_{out} \leftarrow r_{in}$



A problem of more practical interest is the computation of correlations. In this problem pattern, string, and result are all numbers. The result r_i of a correlation is defined as:

$$r_i = (s_{i-k} - p_0)^2 + (s_{i+1-k} - p_1)^2 + \ldots + (s_i - p_k)^2.$$

A good match of substring to pattern results in a high correlation.

Correlations can be computed by a machine with identical data flow to the string matching chip, except that all streams contain numbers. The comparator is replaced by a difference cell that computes:

$$d_{out} \leftarrow s_{in} - p_{in}.$$

This difference computation may be pipelined bitwise in the same way as the character comparison.

An adder cell replaces the accumulator. The algorithm for the adder cell is:

IF λ_{in}
THEN $r_{out} \leftarrow t$; $t \leftarrow 0$
ELSE $r_{out} \leftarrow r_{in}$; $t \leftarrow t + d_{in}^2$.

Many other problems, such as convolutions and FIR filtering, have algorithms that use the same data flow (see [Kung 79b, Kung and Leiserson 79]). It should be clear that special purpose hardware similar to the pattern matching chip can be built for any of these problems.

4. Design Methodology

When designing a complex system of any kind, a systematic approach is essential. The design task must be broken into manageable subtasks, with a well defined flow of information between them. Each subtask can then be performed separately, with no need to consider more than one subtask at a time. This allows division of labor, and, more importantly, prevents mistakes and eases design changes.

Because of the diversity of tasks and concerns in VLSI design, a systematic method is especially important in designing a special purpose chip. It is impossible to take global data flow, circuit design, and transistor characteristics, for instance, into account all at once. We must find small subtasks, with boundaries between them that hide the implementation details of one from another. Of course, any set of subtasks is unlikely to be completely independent, since problems that crop up in performing one of them may require that another subtask be redone. Difficulties in layout, for example, may mandate a circuit redesign, but these design iterations will be easier if the interactions between subtasks are few.

Several natural information boundaries are available in the design of VLSI systems. One advantage of the use of geometrically regular algorithms is the spatial separation that they impose between subsystems. The interior of one cell can be designed in ignorance of the interior details of another (although such exterior details as size and data path positions must be known). If cells are complex, the separation of circuit functions within each cell may provide an additional information boundary. The design of each functional block of a cell can then be largely independent of the others. A further aid to VLSI design is the existence of a hierarchy of abstract models of the chip, from the algorithm level to the gate level to the layout level. Each level of the hierarchy deals with an independent set of design issues, serving as an implementation of the next level up, and a specification of the next level down.

The chip design can thus be decomposed geometrically, functionally, and hierarchically. To use all of these decompositions to their best advantage we must make sure that they are consistent. Tasks that are separated geometrically should also be separated functionally and hierarchically. It would be unfortunate, for example, if all cell circuits had to be considered at once in order to construct a stick diagram for the chip. The way to avoid this is to carefully construct a task dependency graph before beginning the design. This graph should contain all of the subtasks to be performed, together with the information needed for each and the precedence relations among them. Of course, backtrack paths that may result in several iterations of one task because of difficulties in another need not be shown. The chip design task is not yet well enough understood to predict such backtracking.

The purpose of the task dependency graph is to make sure that no more than a small

amount of knowledge is required for any subtask. Each of the subtasks in the graph should deal with the design of one geometric area at one level of abstraction. The circuit design of the entire chip all at once, for example, is too large a task, since it covers too much area. The task of designing a cell stick diagram from its function is too large, since it spans too many levels of the hierarchy. Designing a cell circuit from its function is probably a task of the proper size, although if the cell performs several different functions, the task should be further subdivided. While constructing the task dependency graph, we can make sure that no task is too large, as well as making sure that all information needed for a particular task will be available when that task is performed.

Figure 4-1 is the task dependency graph for the design of the pattern matching chip. It should be suitable for designing chips of about the same scale. The subtasks in the graph were chosen so that each deals with only one geometric region, one circuit function, and one level of the VLSI abstraction hierarchy. The arrows indicate the flow of information between subtasks. In the following we shall briefly describe each of the subtasks. The design of the pattern matching chip described in Section 3.2 followed the steps in this graph, although the graph was not explicitly drawn at the beginning.

Algorithm

The chip design must begin with an algorithm design, which specifies conceptually the overall structure of the chip. In general, several algorithms will exist for any problem, and the best should be found at this stage. The algorithm design is an integrated effort that includes the design of a data flow and geometry for the overall system, as well as a specification of the function of each type of cell. The algorithm is a level of abstraction at which to think about important properties such as regularity and modularity, without worrying about low-level issues. It should supply two distinct bodies of information to the implementation.

One of these is the data flow pattern, including the number of cells, their geometric placements, and the choreography of data. The types of cells should be distinguished, and the beats on which each cell is active should be identified. The other body of information is the function of each cell type. This is not just the circuit function, but also the relative positions of signal inputs and outputs, and the sequence of activity on each beat.

Cell Combinations and Placements

Cells in the implementation might not correspond one to one with the cells in the algorithm. Several cells may be combined to share components, for example, or rarely used communication paths in the algorithm might be multiplexed onto one physical data path. The first task in implementing the algorithm is to decide upon these combinations, and to position

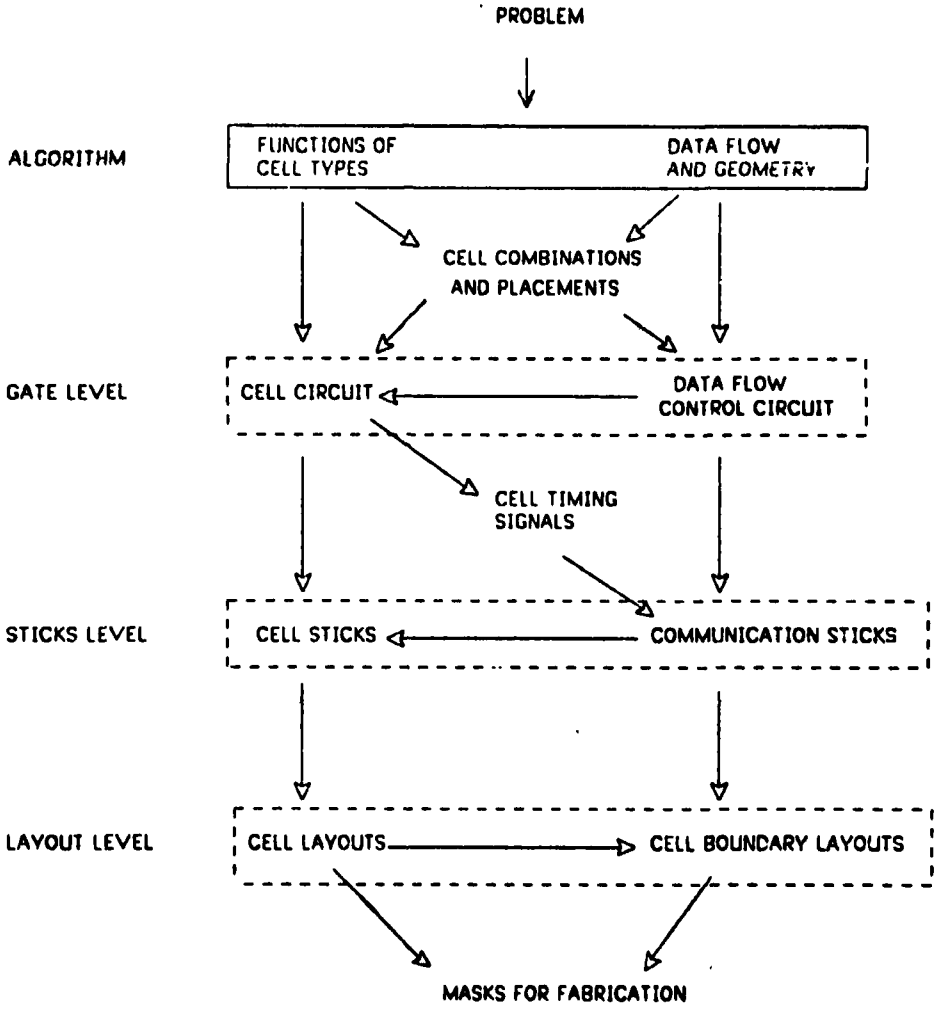


Figure 4-1: Steps in the design of a special purpose chip.

cells and cell combinations upon the chip.

From the algorithm data flow, this task requires the pattern of active and idle cells at each beat, and the use of each communication path. From the cell functions it requires the shareable subfunctions and the complexity of each cell type. The output of the task is a skeleton layout for the chip, with each cell group assigned a location and set of contained cells.

Data Flow Control Circuit

This task is the design of the control circuitry that ensures the orderly movement of data on the correct beats. To perform this task we must learn the correct sequence of beats from the algorithm data flow, along with which elements are active on each beat. From the cell combination task we learn which cell groups and physical data paths contain the active elements.

Based on the size and intended use of the chip we can decide whether the data flow should be clocked or self-timed. If a clock is to be used we decide whether to generate it on the chip or externally. The shift registers for data movement can then be designed, and any clock wires or synchronization signals can be routed among the cell groups on the skeleton layout.

Cell Logic Circuits

We are now in possession of the three pieces of information needed to design circuits for the cells. These are the cell functions (from the algorithm) the group of cells to be implemented by each circuit (from the cell combination step) and the shift register stages that must be included in each cell (from the data flow control). If the cell functions are simple enough, ad hoc techniques of circuit design may be adequate. If the functions are complex, the cells may be split into subsytems to be designed independently. In this way, full advantage may be taken of the functional decomposition of each cell. In addition, the circuit for each cell type can be designed without reference to the others, since all communication needs have been considered in the data flow control. In designing the circuits, consideration must be given to how the chip will be tested after fabrication.

Cell Timing Signals

A cell function may have several distinct steps to be performed in sequence on each beat. In the pattern matching accumulator, for example, the assignments

$$r_{out} \leftarrow t;\ t \leftarrow \text{TRUE}$$

must take place in the correct order. The cell circuit, especially in a clocked system, may require signals to control such sequences, in addition to the signals needed for cell activation. These signals should be supplied by the data flow control. Any such signals should be identified as soon as the cell circuits are all complete, and circuits to generate them added to the data flow control.

Communication Sticks

When the circuitry of the data flow control is complete we can draw its stick diagram. For geometrically regular chips this will consist of an open network of communication path routings, with blank spaces left for the cells, together with some control circuitry. If there is centralized clock circuitry on the chip, its topology can be designed. The distribution network for power and ground should also be designed at this stage.

Cell Sticks

The topological layouts of the individual cells can now be designed. The relative locations of power, ground, and all inputs and outputs are known from the communication sticks. We must now choose implementations for the circuit elements and decide on the relative positions of internal data paths.

Cell Layouts

Once the topological layouts of the cells are complete, the detailed layout of each cell is possible. Following the design rules for the intended fabrication process, actual dimensions for each electrical component and distances between circuit elements must be chosen. The output of this task is a scale drawing of the cell.

Cell Boundary Layouts

With cell sizes known, the cell boundaries can be laid out. The topology of the communication paths and dataflow control is known from the communication sticks. Wire lengths and spacings can be chosen, as can distances between cells. Inputs and outputs can be connected to contact pads, which will be connected to pins on the IC package. The cell boundary layouts and the cell layouts form a complete description of the chip. Once they are complete, masks can be made and the chip can be fabricated.

Summary of the Design Methodology

With the help of the task dependency graph, the seemingly complicated process of designing a special purpose chip can be carried out systematically, one subtask at a time. The graph presented here, although based on limited design experience, seems to be a good starting point. We believe that the design tasks below the algorithm level are relatively routine and may (in principle at least) be helped a great deal by various (future) computer-aided design systems. Eventually the algorithm design level will be the one area requiring substantial effort and experience from a designer.

5. Conclusion

Over the past few years efforts in several fields of computer science have converged to make possible the design of special-purpose chips, as described in this paper. The study of parallel algorithms, particularly those for mesh-connected computers, has provided techniques for VLSI algorithm design (see the survey [Kung 79a]). The work of [Mead and Conway 80] in developing structured techniques for NMOS design has eased the design of reliable circuits and layouts. Improvements in computer-aided design and graphics systems have reduced the drudgery of designing the masks. Finally, the development of suitable intermediate languages makes the design and fabrication processes relatively independent, and allows designs to be shared among several users.

These developments allow the relatively inexperienced designer to develop chips quickly and confidently for his own application. By concentrating on algorithms, chips of good performance and fairly small area can be constructed with minimal design time. The design of the pattern matching chip described in Section 3.2 took only about two man-months. We should see many designs for special purpose chips appearing in the near future.

Further developments can make the task of the designer even easier. It is possible, for example, to build libraries of standard cells, similar to subroutine libraries. If a designer needs, say, an inner product step cell, he may be able to select it from a library rather than construct it himself. Libraries of dataflow implementations are also possible, although their forms are less obvious.

Advances in fabrication technology may increase the scale of projects that can be attempted. Aside from reductions in feature size, the prospect of wafer-scale integration will increase the power of special purpose devices. Modularity of algorithms is especially important in wafer-scale integration, where the circuits on a wafer of silicon are interconnected rather than being cut apart for individual packaging. Manufacturing defects make it essential to be able to modify the interconnections so that a defective circuit is replaced by a functioning one on the same wafer. This can be done easily if there are only a few types of circuits with regular interconnections.

In conclusion, we believe that with the design philosophy and methodology described in the paper the design of special-purpose VLSI chips by their users is practical. Connected to a general-purpose computer, these devices can provide rapid solutions to stressing computations. The time to design special purpose chips has come.

Acknowledgements

We received help from many people during the course of this research. Using a

preliminary version of the text [Mead and Conway 80], Bob Sproull taught us the basic techniques of NMOS design in his VLSI design course at CMU. Our prototypes of the pattern matching chip are included in the XEROX PARC multi-project chips for Spring 1979, and have been fabricated by XEROX facilities. Bob Hon provided much needed help, including building the layout design system we used, converting our designs into ICARUS format and mounting chips for testing. Philip Lehman and Siang Song gave suggestions in the early stages of the algorithm design. Lynn Conway, Bob Hon, Dick Lyon, Siang Song, and Bob Sproull gave us comments on this paper. To these people, and to the others who helped us directly or indirectly, we express our thanks.

References

[Boyer and Moore 77]
Boyer, R. S. and Moore, J. S.
A Fast String Searching Algorithm.

Communications of the ACM 20(10):762, October, 1977.

[Fischer and Paterson 74]
Fischer, M. J. and Paterson, M. S.
String Matching and Other Products.
Technical Report 41, Massachussetts Institute of Technology, Project MAC, 1974.

[Griswold et al. 68]
Griswold, R. E., Poage, J. F., and Polansky, I. P.
The SNOBOL4 Programming Language.
Prentice-Hall, Englewood Cliffs, NJ, 1968.

[Hon and Sequin 79]
Hon, R. and Sequin, C.
A Guide to LSI Implementation.
Technical Report, XEROX Palo Alto Research Center, 1979.
Second Edition.

[Knuth et al. 77]
Knuth, D. E., Morris, J. H., and Pratt, V.R.
Fast Pattern Matching in Strings.
SIAM Journal of Computing 6(2):323-350, June, 1977.

[Kung and Leiserson 79]
Kung, H.T. and Leiserson, C.E.
Systolic Arrays (for VLSI).
In Duff, I. S. and Stewart, G. W., editor, *Sparse Matrix Proceedings 1978*, pages 256-282. Society for Industrial and Applied Mathematics, 1979.
A slightly different version appears in *Introduction to VLSI Systems* by C. A. Mead and L. A. Conway, Addison-Wesley, 1980, Section 8.3.

[Kung 79a]
Kung, H. T.
The Structure of Parallel Algorithms.
In Yovits, M. C., editor, *Advances in Computers, Volume 19.* Academic Press, New York, 1979.

[Kung 79b]
Kung, H.T.
Let's Design Algorithms for VLSI Systems.
In *Proc. Conference on Very Large Scale Integration: Architecture, Design, Fabrication.* California Institute of Technology, January, 1979.
Also available as a CMU Computer Science Department technical report, 1979.

[Mead and Conway 80]
Mead, C.A. and Conway, L.A.

Introduction to VLSI Systems.
Addison-Wesley, Reading, Massachusetts, 1980.

[Mukhopadhyay 79]
Mukhopadhyay, A.
Hardware Algorithms for Nonnumeric Computation.
IEEE Transactions on Computers C-28(6):384-394, June, 1979.

[Seitz 79]
Seitz, C. L.
Self-Timed VLSI Systems.
In *Proc. Conference on Very Large Scale Integration: Architecture, Design, Fabrication.* California Institute of Technology, January, 1979.

[Sutherland and Mead 77]
Sutherland, I.E. and Mead, C.A.
Microelectronics and Computer Science.
Scientific American 237(3):210-228, September, 1977.

[Warter and Mules 79]
Warter, P. J. and Mules, D. W.
A Proposal for an Electronic File Cabinet.
In *Proceedings of Microdelcon.* IEEE, March, 1979.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: (14) CMU-CS-79-147

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): (6) DESIGN OF SPECIAL-PURPOSE VLSI CHIPS: EXAMPLE AND OPINIONS.

5. TYPE OF REPORT & PERIOD COVERED: (9) Interim Report,

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): (10) M.J. Foster & H.T. Kung

8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0370

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Carnegie-Mellon University
Computer Science Department
Pittsburgh, PA 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
Office of Naval Research
Arlington, VA 22217

12. REPORT DATE: (11) Sep [redacted] 79

13. NUMBER OF PAGES: 33

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 12 32

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

(15) N00014-76-C-0370,
F33615-78-C-1551

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

403081